TP10-MNIST-multitask

September 18, 2019

1 Multitask on MNIST

We will use the same MNIST dataset with each example shaped as `(28,28,1)` array.

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt
%matplotlib inline
import datetime as dt

from pathlib import Path

from sklearn.datasets import fetch_openml
from sklearn import preprocessing

import io
import scipy

data_home = '/tmp/scikit_learn_data/'
datafile = '/tmp/mnist.npz'

datapath = Path(datafile)
if not(datapath.exists()):
    print("Data File not found... downloading it")
    Xmnist, ymnist = fetch_openml('mnist_784',
                                  version=1,
                                  return_X_y=True,
                                  data_home=data_home)
    np.savez(datapath.as_posix(),
             X=np.array(Xmnist, dtype='u8'),
             y=np.array(ymnist, dtype='u8'))
    print("Data File downloaded and saved")
    del Xmnist, ymnist

print("Data File found... loading it into memory")
data = np.load(datapath.as_posix())
```

```
Xmnist = data['X']/255.
ymnist = keras.utils.to_categorical(data['y'])
print("Data File loaded")

Xtrain, Ytrain = Xmnist[:50000], ymnist[:50000]
Xvalid, Yvalid = Xmnist[50000:60000], ymnist[50000:60000]
Xtest, Ytest = Xmnist[-10000:], ymnist[-10000:]

Xtrain = Xtrain.reshape((Xtrain.shape[0], 28, 28, 1))
Xvalid = Xvalid.reshape((Xvalid.shape[0], 28, 28, 1))
Xtest = Xtest.reshape((Xtest.shape[0], 28, 28, 1))
```

```
Data File found... loading it into memory
Data File loaded
```

1.1 Model sharing

Let's say that you want to do two classification tasks on the same data set but with different classes. For example, the first task can be to guess the nature of the object in an image (task A) and the second task to guess the color of the object (task B). In this setup it is interesting to share the lower layers of both tasks.

Keras let you compose models into other models. Thus layers sharing can easily be done like this:

```
# Let's build the basement layers
# Sequential API
basement = keras.Sequential()
basement.add([...])
basement.add([...])
# or Functional API
x = keras.Input([...])
h = somefunc(x)
y = somefunc(h)
basement = keras.Model(inputs=x, outputs=y)

# Then basement can be used as a operator for building other models

# Using the sequential API
taskA = keras.Sequential()
taskA.add(basement)
taskA.add([...])

# or using the functional API
x = keras.Input([...])
h = bassement(x)
y = someotherfunc(h)
taskB = keras.Model(inputs=x, outputs=y)
```

The weights of the `basement` layers are shared between `taskA` and `taskB`. If you fit `taskA` this will pre-trained `taskB`.

1.2 Multi-tasking

Let's say that you have two models, A and B, sharing some input, weights or not and you want to train them at the time. You can easily to this by building a multi-tasking, multi-output auxiliary model.

```
# Task A
x = keras.Input([...])
#[...]
y = ...
taskA = keras.Model(inputs=x, outputs=y, name='taska') # IMPORTANT you must name your model

# Task B
x = keras.Input([...])
#[...]
y = ...
taskB = keras.Model(inputs=x, outputs=y, name='taskb') # IMPORTANT you must name your model

# MultiTask Model
x = keras.Input([...])
ya = taskA(x)
yb = taskB(x)
multitask = keras.Model(inputs=x, outputs=[ya,yb])

# Now you can replace compile and fit arguments when they are specific to each model by a dict

multitask.compile(
    optimizer=keras.optimizers.Adam(lr=1e-3),
    loss={'taska': 'categorical_crossentropy', 'taskb: 'mean_squared_error'},
    loss_weights={'taska': 1., 'taskb': 0.2}, # Important: task weighting !
    metrics={'taska': ['categorical_accuracy'], 'taskb':['mean_squared_error']},
)

multitask.fit(Xtrain,
                {'taska': YtrainA, 'taskb': YtrainB},  #
                epochs=20, batch_size=batch_size,
                validation_data=(Xvalid, {'taska': YvalidA, 'taskb': YvalidB}),
                )
```

1.3 Exercise

We have done classification and auto-encoding on MNIST...

Let's build a classifier that's using the encoder as its first layers. Let's train the classifier and the auto-encoder at the same time !